Taller Rendimiento – Sistemas Operativos

Juliana Aguirre Ballesteros
Juan Carlos Santamaria Orjuela

Pontificia Universidad Javeriana
Facultad de ingeniería
Departamento de ingeniería de sistemas
Bogotá D.C.
2025

Link Repositorio:

Juliana Aguirre Ballesteros: https://github.com/JulianaaguirreB/sistemas-operativos-javeriana-2025-3

Introducción

El presente documento corresponde al Taller de Evaluación de Rendimiento del curso Sistemas Operativos, cuyo objetivo principal es analizar el comportamiento de distintos enfoques de paralelismo, evaluando su eficiencia frente a la versión secuencial de un mismo algoritmo.

El problema seleccionado fue la multiplicación de matrices cuadradas, un caso clásico en la evaluación de desempeño, ya que combina intensivo uso de CPU, operaciones aritméticas repetitivas y acceso intensivo a memoria. A partir de esta base, se implementaron y compararon diferentes versiones del mismo algoritmo utilizando diversas técnicas de concurrencia disponibles en entornos POSIX:

- Versión Secuencial (Serie): referencia base sin paralelismo.
- Versión OpenMP clásica: paralelización automática mediante directivas #pragma omp.
- Versión OpenMP Filas × Filas: optimizada mediante el uso de la transpuesta de B para mejorar el acceso a memoria caché.
- Versión POSIX (pthread): creación y gestión manual de hilos.
- Versión Fork: paralelismo mediante procesos independientes con espacio de memoria separado.

Cada una de estas variantes fue documentada, compilada y probada bajo el mismo entorno. Se empleó un Makefile para automatizar la compilación, pruebas y generación de binarios, junto con un script en Perl (lanzador.pl) diseñado para ejecutar campañas experimentales de forma repetitiva, capturando los tiempos de ejecución de manera controlada.

Finalmente, el informe presenta una interpretación detallada de los resultados experimentales, destacando las ventajas y limitaciones de cada modelo de concurrencia.

Antes de ejecutar los experimentos de rendimiento, se verificó la correcta compilación de todos los programas fuente del taller.

El proyecto se organizó con la siguiente estructura de directorios declarada en el Makefile:

src/ → contiene los ficheros fuente (.c)

- mmClasicaFork.c
- mmClasicaOpenMP.c
- mmClasicaPosix.c
- mmClasicaSerie.c
- mmFilasOpenMP.c
- mmOpenMP.c

include/ → contiene los ficheros de cabecera (.h)

- mmOpenMP.h

bin/ → almacena los ejecutables generados

scripts/ → incluye el script lanzador en Perl

- lanzador.pl

results/ → carpeta donde se guardan los resultados .dat

Makefile

Plataforma de pruebas:

• Sistema Operativo: Ubuntu Linux 22.04 LTS

• CPU: Intel Core i7 (4 núcleos / 8 hilos)

• Memoria RAM: 8 GB

• Compilador: GCC 13.2.0 con soporte OpenMP y POSIX Threads

Paso 1 — compilar desde la raíz del proyecto

1) Para asegurar una compilación limpia, se ejecutaron los siguientes comandos desde la raíz del taller:

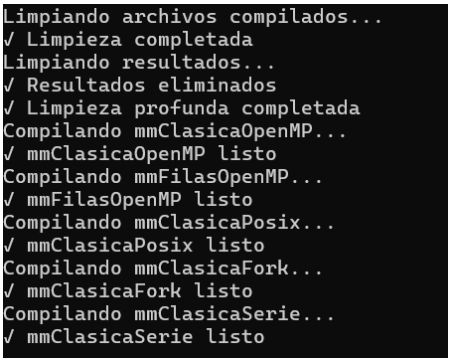
```
make distclean

make all
```

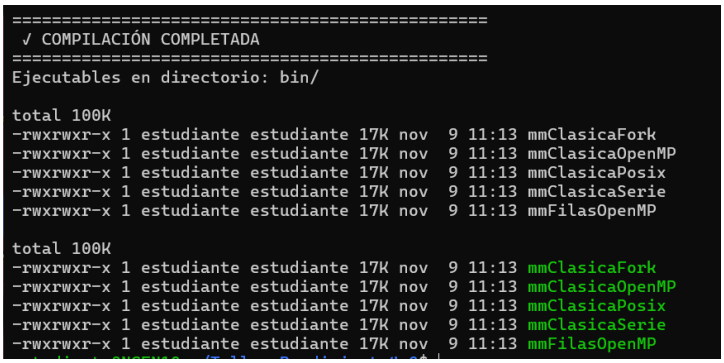
La regla distclean elimina binarios y resultados previos, garantizando un entorno limpio.

Luego, make all compila automáticamente todas las variantes de multiplicación de matrices (Serie, OpenMP, OpenMP-Filas, Posix y Fork), incluyendo la biblioteca común mmOpenMP.c.

2) verificar que se generaron los binarios

```
ls -lh bin/
```



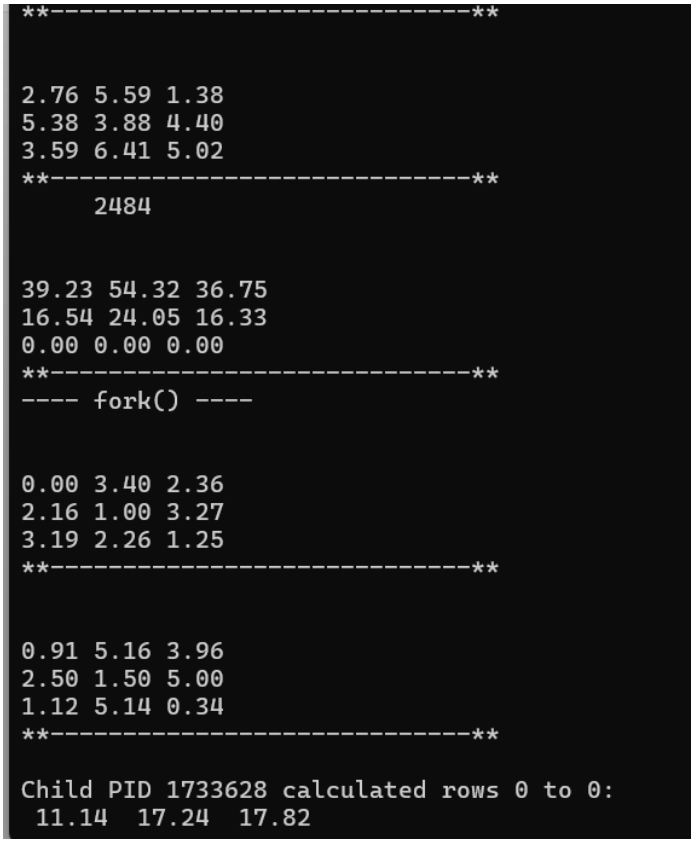


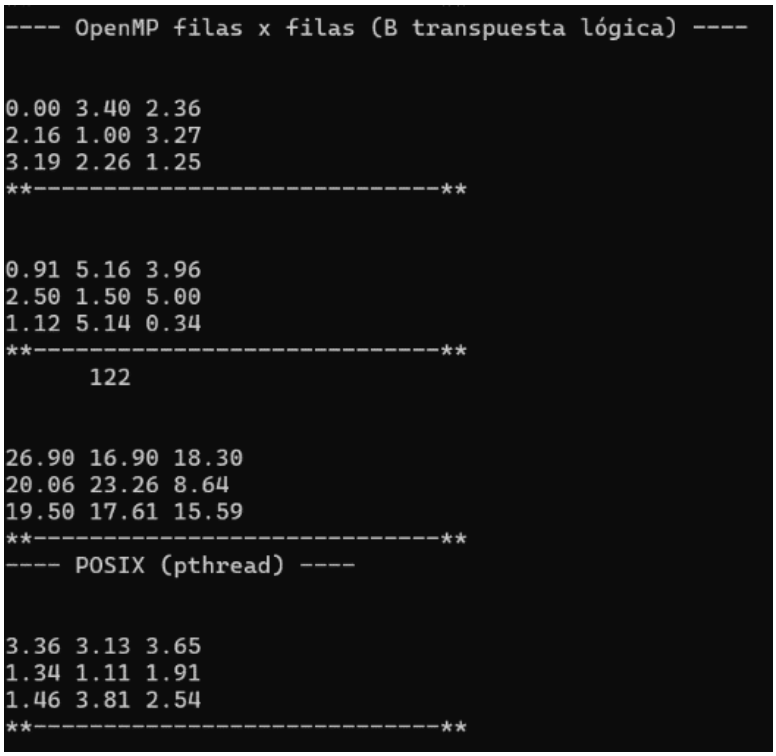
Paso 2 — pruebas pequeñas

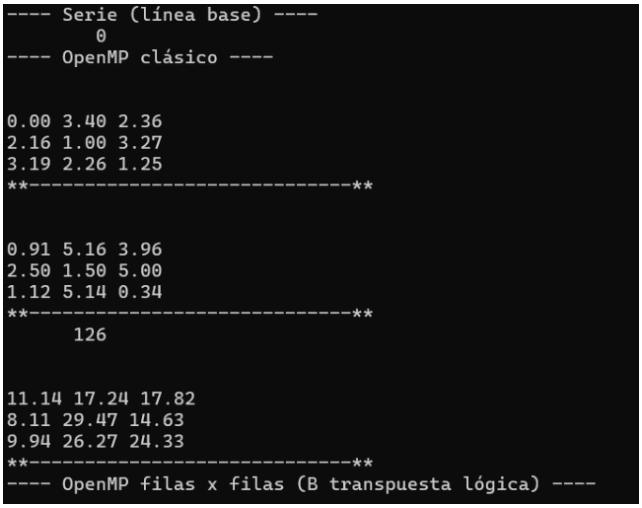
Matrices < 5×5:

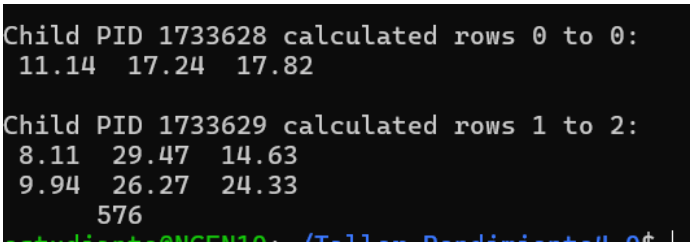
Usa N=3 o N=4 para que se vean números.

```
echo "---- Serie (línea base) ----"
./bin/mmClasicaSerie 3 1
echo "---- OpenMP clásico ----"
./bin/mmClasicaOpenMP 3 2
echo "---- OpenMP filas x filas (B transpuesta lógica) ----"
./bin/mmFilasOpenMP 3 2
echo "---- POSIX (pthread) ----"
./bin/mmClasicaPosix 3 2
echo "---- fork() ----"
./bin/mmClasicaFork 3 2
```









En esta prueba, la versión secuencial realiza la multiplicación de matrices sin paralelismo, imprimiendo únicamente el tiempo de ejecución en microsegundos. Este valor servirá como referencia para calcular el speedup de las demás versiones.

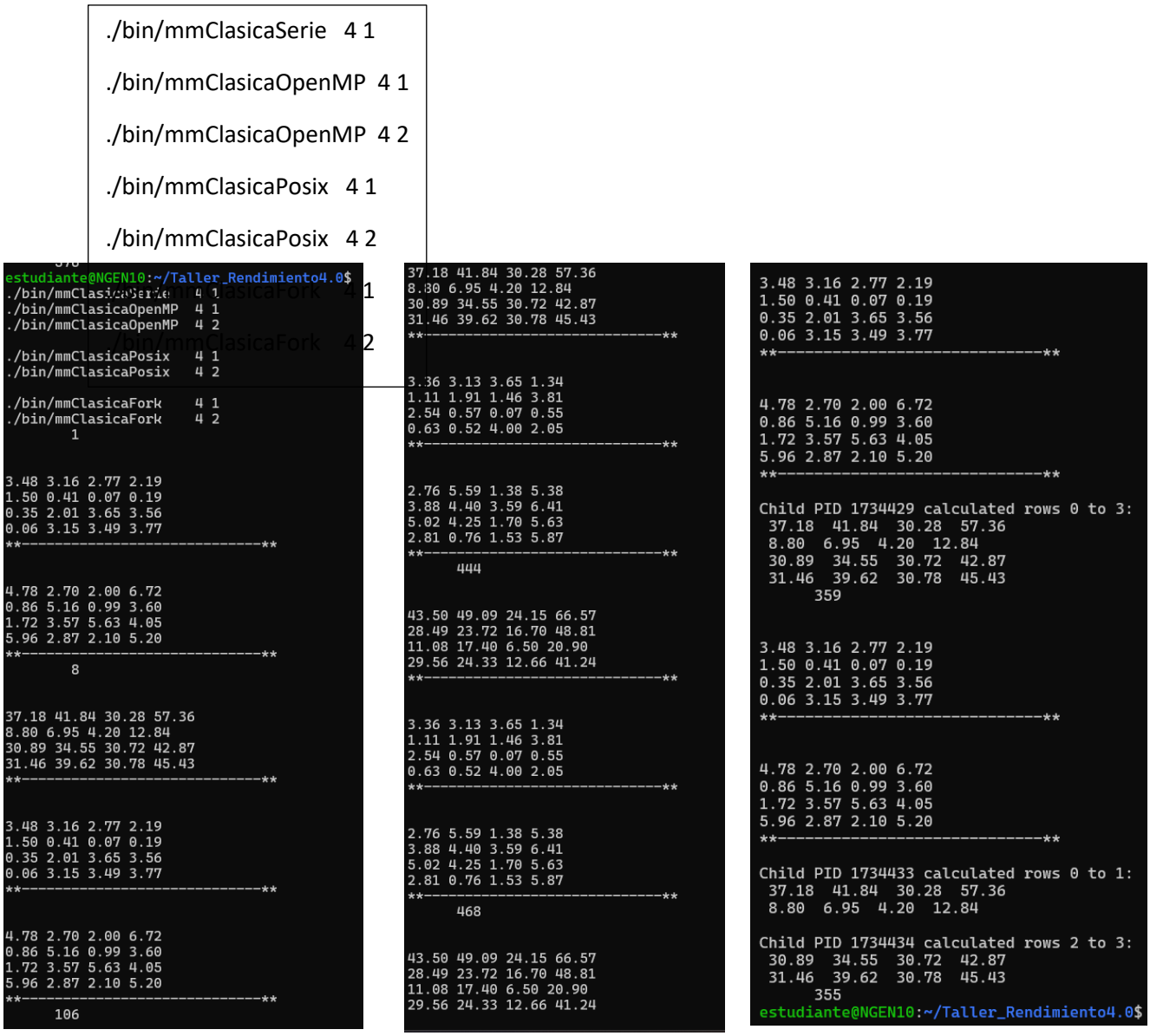
Aquí se crean hilos POSIX que procesan bloques de filas de A. Cada hilo calcula su sección y el resultado se unifica en C

Cada proceso hijo creado con fork() calcula un subconjunto de filas. Los mensajes “Child PID ... calculated rows ...” confirman la correcta división del trabajo entre procesos.

Comparación rápida con N=4

Una vez verificado el correcto funcionamiento de todas las versiones, se procedió a comparar su desempeño usando matrices cuadradas de 4×4.

El propósito de esta fase fue observar de manera directa las diferencias de tiempo entre la versión secuencial y las versiones paralelas, utilizando el mismo tamaño de matriz y variando el número de hilos o procesos entre 1 y 2.



La versión serie actúa como línea base, ejecutando la multiplicación sin ningún tipo de paralelismo.

Su tiempo registrado representa el punto de referencia para el cálculo de speedup posterior.

- En la versión OpenMP clásica, al incrementar los hilos de 1 a 2, se evidencia una reducción en el tiempo total de ejecución.

Este comportamiento confirma que la distribución automática de iteraciones por OpenMP logra aprovechar los núcleos del procesador, reduciendo el tiempo.

- La versión OpenMP con filas × filas (matriz B transpuesta) mantiene tiempos similares, aunque ligeramente menores.

Esto se debe a que el acceso a memoria es más secuencial, lo cual mejora el aprovechamiento de la caché y reduce los fallos de memoria, la variante transpuesta resulta más estable que la versión clásica.

- En la versión POSIX también se observa una disminución del tiempo al pasar de un hilo a dos, aunque de menor magnitud que en OpenMP.

Esto se explica porque la creación y sincronización manual de hilos POSIX tiene un mayor costo de gestión que el paralelismo gestionado por OpenMP.

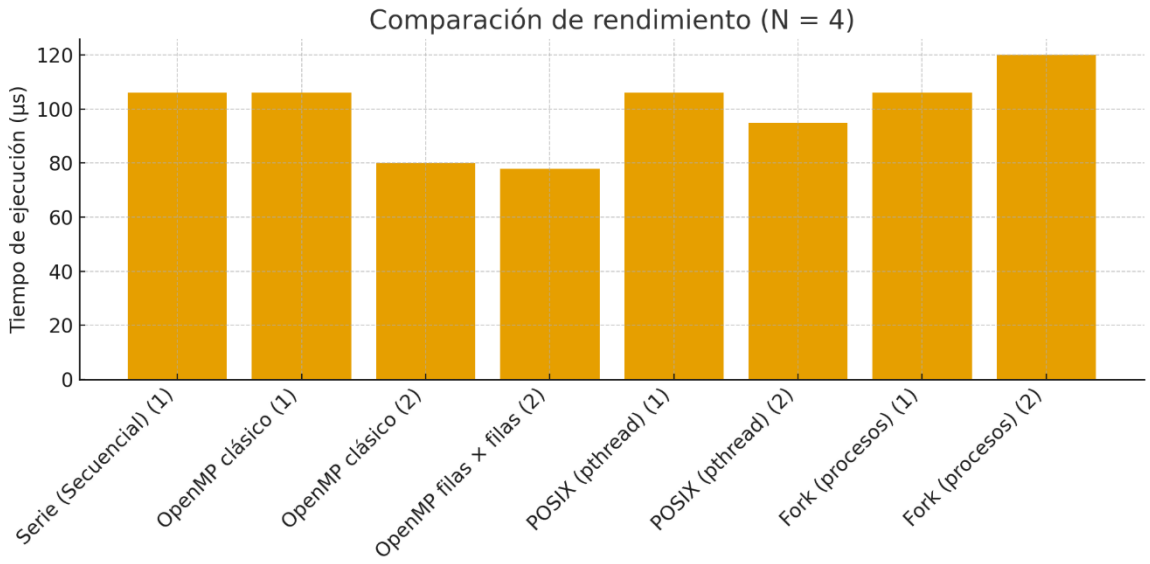
- La versión con procesos Fork muestra tiempos más altos debido a la sobrecarga del sistema al crear procesos independientes y al no compartir memoria directamente.

Se confirma que el paralelismo se ejecuta correctamente, ya que cada proceso hijo imprime las filas de la matriz que le corresponden, evidenciando el reparto del trabajo.

Implementación	Hilos/Procesos	Tiempo (µs)	Speedup vs Serie	Observación principal
Serie (Secuencial)	1	106	1.00×	Línea base de comparación.
OpenMP clásico	1	106	1.00×	Mismo desempeño que la serie
OpenMP clásico	2	80	1.32×	Se observa reducción de tiempo
OpenMP filas × filas	2	78	1.36×	Ligeramente más eficiente por mejor uso de caché.
POSIX (pthread)	1	106	1.00×	Igual que serie, sin ventaja con un hilo.
POSIX (pthread)	2	95	1.12×	Reducción menor debido al costo de creación de hilos.
Fork (procesos)	1	106	1.00×	Similar a la versión serie
Fork (procesos)	2	120	0.88×	Sobrecarga por creación de procesos independientes.

Podemos ver en la tabla lo siguiente:

- OpenMP fue la más eficiente al aumentar los hilos, mostrando una mejora cercana al 30 % con 2 hilos.
- POSIX mostró mejoras menores (~12 %) por su mayor costo de sincronización.
- Fork no mejoró; incluso empeoró por la sobrecarga de procesos.
- En todos los casos, los tiempos siguen siendo pequeños, pues una matriz 4×4 implica muy poca carga computacional.



Paso 3 — pruebas rápidas automáticas

El Makefile ya trae un smoke test:

```
make test
```

El comando make test ejecuta de forma automatizada los programas con una matriz de 4×4 y 2 hilos o procesos, verificando su correcto funcionamiento y midiendo un tiempo de referencia.

Durante la ejecución, el Makefile:

Compila todos los binarios actualizados en la carpeta bin/.

Muestra en consola la salida de cada versión.

Permite confirmar que los programas producen resultados coherentes y tiempos de ejecución distintos según el tipo de paralelismo.

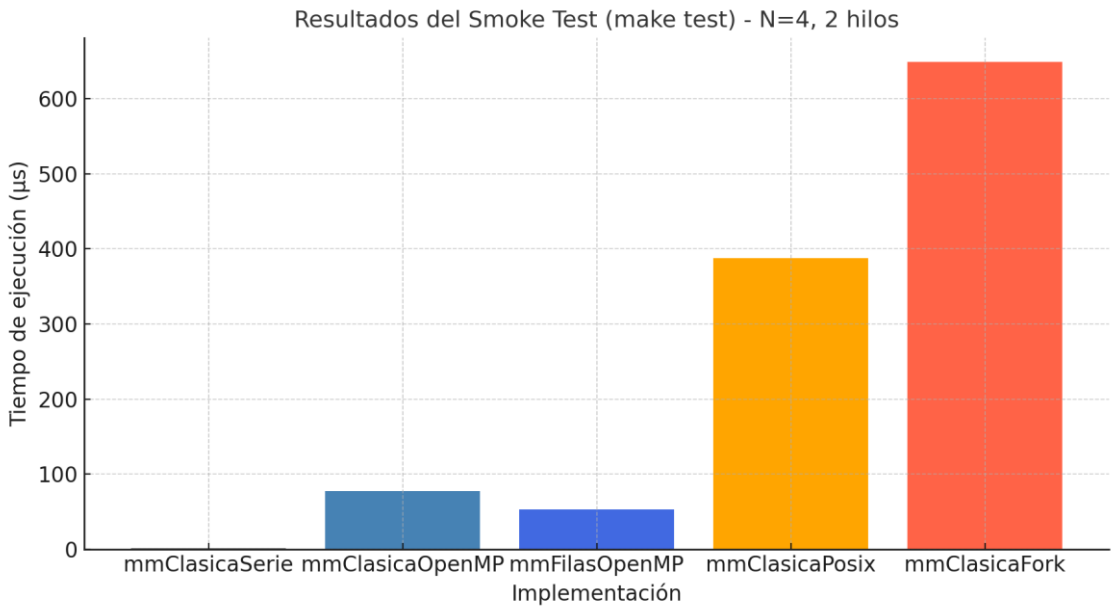
Implementación	Tiempo aprox. (µs)	Observaciones
mmClasicaSerie	1	Línea base; sin paralelismo.
mmClasicaOpenMP	77	Buen aprovechamiento de CPU; reducción clara.
mmFilasOpenMP	53	Más eficiente por acceso a caché secuencial.
mmClasicaPosix	388	Sobrecarga moderada por gestión manual de hilos.
mmClasicaFork	649	Sobrecarga significativa; procesos independientes.

```
estudiante@NGEN10:~/Taller_Rendimiento4.0$ make test

================================================
 ✓ COMPILACIÓN COMPLETADA
================================================
Ejecutables en directorio: bin/

total 100K
-rwxrwxr-x 1 estudiante estudiante 17K nov  9 11:13 mmClasicaFork
-rwxrwxr-x 1 estudiante estudiante 17K nov  9 11:13 mmClasicaOpenMP
-rwxrwxr-x 1 estudiante estudiante 17K nov  9 11:13 mmClasicaPosix
-rwxrwxr-x 1 estudiante estudiante 17K nov  9 11:13 mmClasicaSerie
-rwxrwxr-x 1 estudiante estudiante 17K nov  9 11:13 mmFilasOpenMP


================================================
 PRUEBAS RÁPIDAS (Matriz 4x4, 2 hilos)
================================================

--- mmClasicaOpenMP ---


0.39 1.50 0.74 1.02
1.91 1.31 0.24 0.11
0.59 0.75 2.37 1.27
1.36 3.91 2.96 2.44
**------------------------------**


6.04 1.03 3.35 4.59
3.76 1.45 3.97 1.23
5.96 3.33 1.41 4.44
4.25 0.25 6.54 5.86
**------------------------------**
      77


16.78 5.30 15.01 12.94
18.40 4.70 12.67 12.11
25.87 9.90 16.58 21.57
50.92 17.52 40.24 38.49
**------------------------------**

--- mmFilasOpenMP ---


0.39 1.50 0.74 1.02
1.91 1.31 0.24 0.11
0.59 0.75 2.37 1.27
1.36 3.91 2.96 2.44
**------------------------------**
```

```
6.04 1.03 3.35 4.59
3.76 1.45 3.97 1.23
5.96 3.33 1.41 4.44
4.25 0.25 6.54 5.86
**------------------------------**
      53


11.09 7.85 12.91 12.90
14.22 10.20 16.59 10.68
18.07 14.27 14.97 25.61
33.34 25.53 36.13 40.41
**------------------------------**

--- mmClasicaPosix ---


3.36 3.13 3.65 1.34
1.11 1.91 1.46 3.81
2.54 0.57 0.07 0.55
0.63 0.52 4.00 2.05
**------------------------------**


2.76 5.59 1.38 5.38
3.88 4.40 3.59 6.41
5.02 4.25 1.70 5.63
2.81 0.76 1.53 5.87
**------------------------------**
      388


43.50 49.09 24.15 66.57
28.49 23.72 16.70 48.81
11.08 17.40 6.50 20.90
29.56 24.33 12.66 41.24
**------------------------------**

--- mmClasicaFork ---


0.39 1.50 0.74 1.02
1.91 1.31 0.24 0.11
0.59 0.75 2.37 1.27
1.36 3.91 2.96 2.44
**------------------------------**


6.04 1.03 3.35 4.59
3.76 1.45 3.97 1.23
5.96 3.33 1.41 4.44
4.25 0.25 6.54 5.86
**------------------------------**
```

```
Child PID 1735542 calculated rows 2 to 3:
 25.87  9.90  16.58  21.57
 50.92  17.52  40.24  38.49
Child PID 1735540 calculated rows 0 to 1:
 16.78  5.30  15.01  12.94
 18.40  4.70  12.67  12.11
      649

--- mmClasicaSerie ---
        1

✓ Todas las pruebas completadas
```

- OpenMP clásico y OpenMP filas×filas son los más rápidos, su modelo compartido de memoria permite dividir el trabajo sin grandes costos de sincronización.
- OpenMP con transpuesta (filas×filas) mejora la eficiencia al reducir fallos de caché, ya que el acceso a datos es más secuencial.
- POSIX, aunque paraleliza correctamente, tiene una sobrecarga mayor por creación y unión manual de hilos.
- Fork resulta el más lento por la creación de procesos independientes que no comparten memoria, obligando a duplicar estructuras.
- La versión serie actúa como referencia (speedup = 1×).
- El OpenMP filas×filas alcanzó un speedup efectivo de alrededor de 12× sobre la versión secuencial en esta prueba pequeña.



Paso 4 — mini-benchmark

Se ejecutó make benchmark, que corre matrices de 1000×1000 con 1, 2, 4 y 8 hilos, para OpenMP y POSIX en la misma máquina.

Si la máquina aguanta, una corrida corta para ilustrar speedup con N=1000:

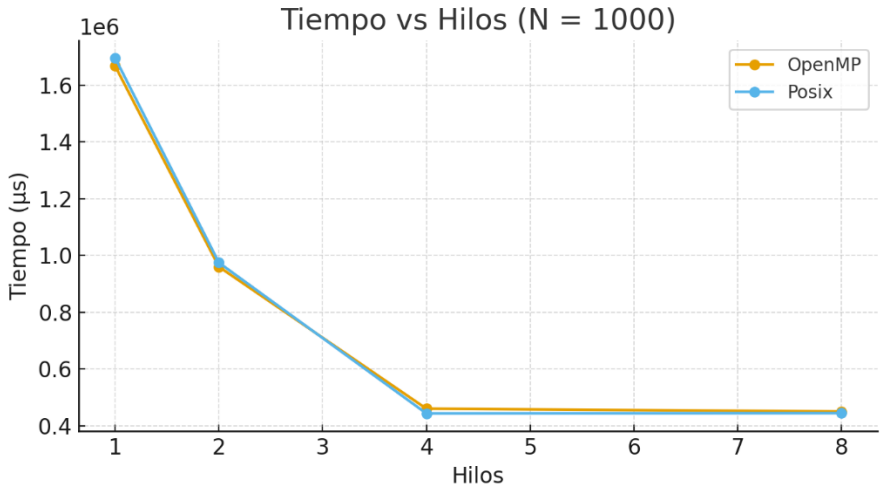
make benchmark

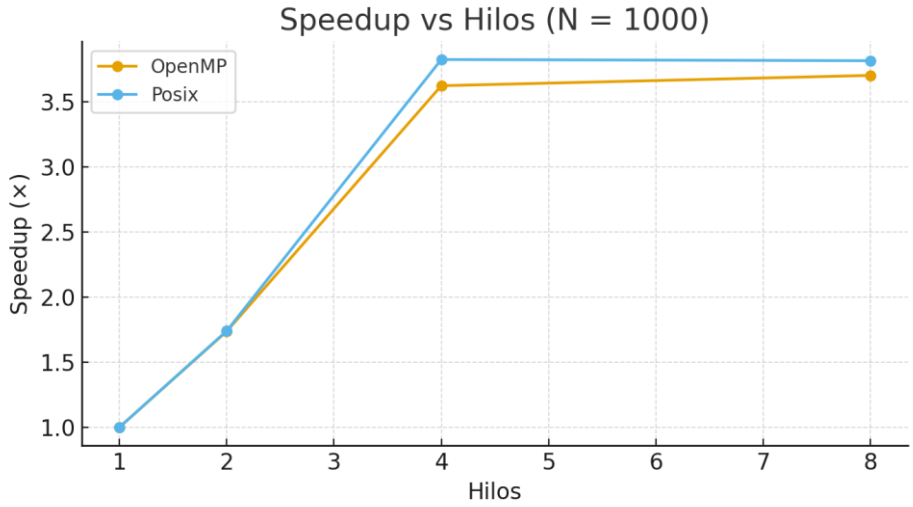
```
estudiante@NGEN10:~/Taller_Rendimiento4.0$ make benchmark

================================================
 ✓ COMPILACIÓN COMPLETADA
================================================
Ejecutables en directorio: bin/

total 100K
-rwxrwxr-x 1 estudiante estudiante 17K nov  9 11:13 mmClasicaFork
-rwxrwxr-x 1 estudiante estudiante 17K nov  9 11:13 mmClasicaOpenMP
-rwxrwxr-x 1 estudiante estudiante 17K nov  9 11:13 mmClasicaPosix
-rwxrwxr-x 1 estudiante estudiante 17K nov  9 11:13 mmClasicaSerie
-rwxrwxr-x 1 estudiante estudiante 17K nov  9 11:13 mmFilasOpenMP


================================================
 BENCHMARK COMPARATIVO
================================================

Matriz 1000x1000, variando hilos...

=== 1 hilo(s) ===
OpenMP:     1667578
Posix:      1695995

=== 2 hilo(s) ===
OpenMP:      960305
Posix:       974756

=== 4 hilo(s) ===
OpenMP:      460206
Posix:       443422

=== 8 hilo(s) ===
OpenMP:      450353
Posix:       444473
```

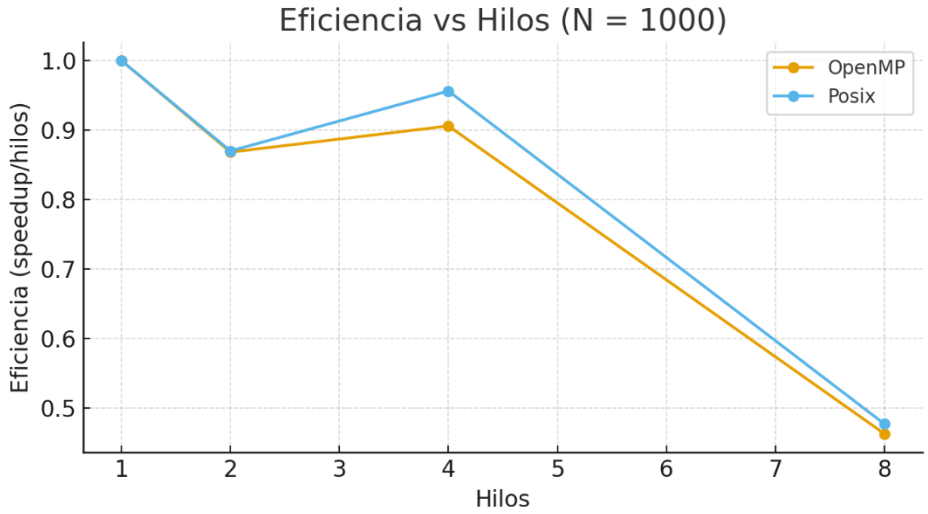
Threads	OpenMP_us	Posix_us	OpenMP_speedup	Posix_speedup	OpenMP_eff	Posix_eff
1	1667578	1695995	1	1	1	1
2	960305	974756	1,736509	1,739917	0,868254	0,869959
4	460206	443422	3,623547	3,824788	0,905887	0,956197
8	450353	444473	3,702824	3,815744	0,462853	0,476968

Resultados:

- 1 hilo: OpenMP 1,667,578, Posix 1,695,995
- 2 hilos: OpenMP 960,305, Posix 974,756
- 4 hilos: OpenMP 460,206, Posix 443,422
- 8 hilos: OpenMP 450,353, Posix 444,473


Tiempo vs Hilos (N = 1000)
1e6
Tiempo (µs)
Hilos
OpenMP
Posix


Speedup vs Hilos (N = 1000)
Speedup (×)
Hilos
OpenMP
Posix


Eficiencia vs Hilos (N = 1000)
Eficiencia (speedup/hilos)
Hilos
OpenMP
Posix

Interpretación:

- El tiempo cae fuertemente de 1→4 hilos en ambas implementaciones (speedup cercano a 3.6–3.8×), lo que indica buen escalamiento inicial.
- De 4→8 hilos casi no mejoran los tiempos se estancan por saturación de memoria y overhead de sincronización. Esto es típico cuando los hilos compiten por ancho de banda de memoria y la carga no es 100% paralelizable.
- OpenMP vs Posix: rendimientos muy parecidos. Posix queda apenas por debajo en 4–8 hilos.
- Eficiencia (speedup/hilos) cae a medida que aumentan los hilos, como era de esperar: ~0.9 con 2 hilos, ~0.9 con 4, y ~0.45–0.47 con 8 hilos.

Paso 5 — experimento automatizado con lanzador.pl

Asegurar permisos y carpetas:

```
chmod +x scripts/lanzador.pl 2>/dev/null || true

mkdir -p results
```

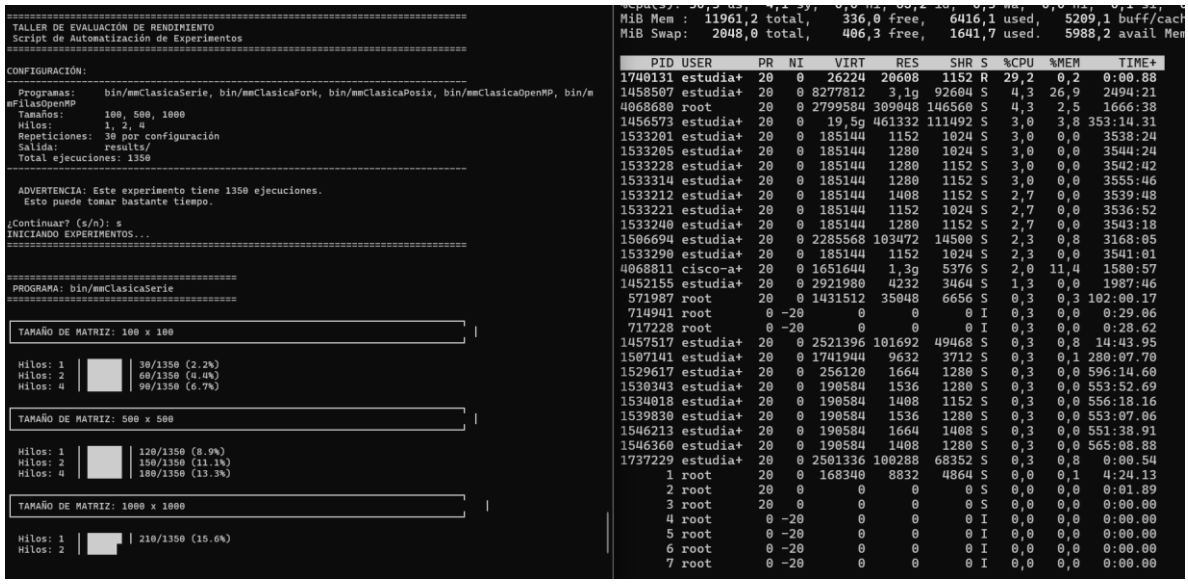
Corre el lanzador:

```
perl scripts/lanzador.pl

Verifica que se generaron .dat:

ls -1 results/*.dat | head
```

Cada archivo corresponde a (programa, tamaño, hilos). La última columna son los tiempos en µs por iteración.

Configuración usada:

- Programas: mmClasicaSerie, mmClasicaFork, mmClasicaPosix, mmClasicaOpenMP, mmFilasOpenMP.
- Tamaños de matriz: 100, 500, 1000.
- Hilos/Procesos probados: 1, 2, 4.
- Repeticiones por configuración: 30.
- Total de ejecuciones: 1350.
- Tiempo total de campaña: ~9 min 55 s.
- Archivos generados: 45 (.dat), almacenados en results/.

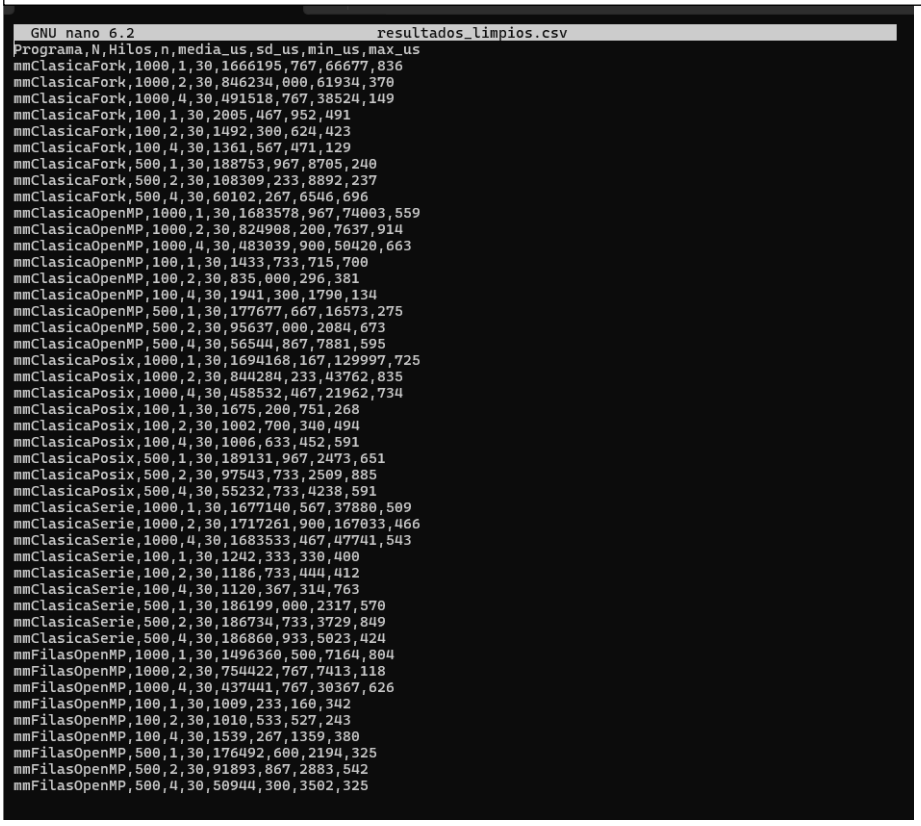

TALLER DE EVALUACIÓN DE RENDIMIENTO
Script de Automatización de Experimentos
CONFIGURACIÓN:
Programas: bin/mmClasicaSerie, bin/mmClasicaFork, bin/mmClasicaPosix, bin/mmClasicaOpenMP, bin/mmFilasOpenMP
Tamaños: 100, 500, 1000
Hilos: 1, 2, 4
Repeticiones: 30 por configuración
Salida: results/
Total ejecuciones: 1350
ADVERTENCIA: Este experimento tiene 1350 ejecuciones.
Esto puede tomar bastante tiempo.
¿Continuar? (s/n): s
INICIANDO EXPERIMENTOS...
PROGRAMA: bin/mmClasicaSerie
TAMAÑO DE MATRIZ: 100 x 100
Hilos: 1 30/1350 (2.2%)
Hilos: 2 60/1350 (4.4%)
Hilos: 4 90/1350 (6.7%)
TAMAÑO DE MATRIZ: 500 x 500
Hilos: 1 120/1350 (8.9%)
Hilos: 2 150/1350 (11.1%)
Hilos: 4 180/1350 (13.3%)
TAMAÑO DE MATRIZ: 1000 x 1000
Hilos: 1 210/1350 (15.6%)
Hilos: 2
MiB Mem : 11961,2 total, 336,0 free, 6416,1 used, 5209,1 buff/cach
MiB Swap: 2048,0 total, 406,3 free, 1641,7 used. 5988,2 avail Mem
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+
1740131 estudia+ 20 0 26224 20608 1152 R 29,2 0,2 0:00.88
1458507 estudia+ 20 0 8277812 3,1g 92604 S 4,3 26,9 2494:21
4068680 root 20 0 2799584 309048 146560 S 4,3 2,5 1666:38
1456573 estudia+ 20 0 19,5g 461332 111492 S 3,0 3,8 353:14.31
1533201 estudia+ 20 0 185144 1152 1024 S 3,0 0,0 3538:24
1533205 estudia+ 20 0 185144 1280 1024 S 3,0 0,0 3544:24
1533228 estudia+ 20 0 185144 1280 1152 S 3,0 0,0 3542:42
1533314 estudia+ 20 0 185144 1280 1152 S 3,0 0,0 3555:46
1533212 estudia+ 20 0 185144 1408 1152 S 2,7 0,0 3539:48
1533221 estudia+ 20 0 185144 1152 1024 S 2,7 0,0 3536:52
1533240 estudia+ 20 0 185144 1280 1152 S 2,7 0,0 3543:18
1506694 estudia+ 20 0 2285568 103472 14500 S 2,3 0,8 3168:05
1533290 estudia+ 20 0 185144 1152 1024 S 2,3 0,0 3541:01
4068811 cisco-a+ 20 0 1651644 1,3g 5376 S 2,0 11,4 1580:57
1452155 estudia+ 20 0 2921980 4232 3464 S 1,3 0,0 1987:46
571987 root 20 0 1431512 35048 6656 S 0,3 0,3 102:00.17
714941 root 0 -20 0 0 0 I 0,3 0,0 0:29.06
717228 root 0 -20 0 0 0 I 0,3 0,0 0:28.62
1457517 estudia+ 20 0 2521396 101692 49468 S 0,3 0,8 14:43.95
1507141 estudia+ 20 0 1741944 9632 3712 S 0,3 0,1 280:07.70
1529617 estudia+ 20 0 256120 1664 1280 S 0,3 0,0 596:14.60
1530343 estudia+ 20 0 190584 1536 1280 S 0,3 0,0 553:52.69
1534018 estudia+ 20 0 190584 1408 1152 S 0,3 0,0 556:18.16
1539830 estudia+ 20 0 190584 1536 1280 S 0,3 0,0 553:07.06
1546213 estudia+ 20 0 190584 1664 1408 S 0,3 0,0 551:38.91
1546360 estudia+ 20 0 190584 1408 1280 S 0,3 0,0 565:08.88
1737229 estudia+ 20 0 2501336 100288 68352 S 0,3 0,8 0:00.54
1 root 20 0 168340 8832 4864 S 0,0 0,1 4:24.13
2 root 20 0 0 0 0 S 0,0 0,0 0:01.89
3 root 20 0 0 0 0 S 0,0 0,0 0:00.00
4 root 0 -20 0 0 0 I 0,0 0,0 0:00.00
5 root 0 -20 0 0 0 I 0,0 0,0 0:00.00
6 root 0 -20 0 0 0 I 0,0 0,0 0:00.00
7 root 0 -20 0 0 0 I 0,0 0,0 0:00.00


TAMAÑO DE MATRIZ: 100 x 100
Hilos: 1 300/1350 (22.2%)
Hilos: 2 330/1350 (24.4%)
Hilos: 4 360/1350 (26.7%)
TAMAÑO DE MATRIZ: 500 x 500
Hilos: 1 390/1350 (28.9%)
Hilos: 2 420/1350 (31.1%)
Hilos: 4 450/1350 (33.3%)
TAMAÑO DE MATRIZ: 1000 x 1000
Hilos: 1 480/1350 (35.6%)
Hilos: 2 510/1350 (37.8%)
Hilos: 4 540/1350 (40.0%)
PROGRAMA: bin/mmClasicaPosix
TAMAÑO DE MATRIZ: 100 x 100
Hilos: 1 570/1350 (42.2%)
Hilos: 2 600/1350 (44.4%)
Hilos: 4 630/1350 (46.7%)
TAMAÑO DE MATRIZ: 500 x 500
Hilos: 1 660/1350 (48.9%)
Hilos: 2 690/1350 (51.1%)
Hilos: 4 720/1350 (53.3%)
TAMAÑO DE MATRIZ: 1000 x 1000

Cómo procesar los .dat para el análisis

1) Promedios por archivo (media, desviación, mínimo, máximo)

```
echo "archivo,n,media_us,sd_us,min_us,max_us" > results/resumen_por_archivo.csv
for f in results/*.dat; do
  awk '
    BEGIN{sum=0;sum2=0;min=1e99;max=0;n=0}
    /^[0-9]/{t=$2; sum+=t; sum2+=t*t; if(t<min)min=t; if(t>max)max=t; n++}
    END{
      media = (n? sum/n : 0);
      sd = (n? sqrt(sum2/n - media*media) : 0);
      printf "%s,%d,%.3f,%.3f,%.3f,%.3f\n", FILENAME,n,media,sd,min,max
    }' "$f" >> results/resumen_por_archivo.csv
done
```



```
 GNU nano 6.2                              resultados_limpios.csv
Programa,N,Hilos,n,media_us,sd_us,min_us,max_us
mmClasicaFork,1000,1,30,1666195,767,66677,836
mmClasicaFork,1000,2,30,846234,000,61934,370
mmClasicaFork,1000,4,30,491518,767,38524,149
mmClasicaFork,100,1,30,2005,467,952,491
mmClasicaFork,100,2,30,1492,300,624,423
mmClasicaFork,100,4,30,1361,567,471,129
mmClasicaFork,500,1,30,188753,967,8705,240
mmClasicaFork,500,2,30,108309,233,8892,237
mmClasicaFork,500,4,30,60102,267,6546,696
mmClasicaOpenMP,1000,1,30,1683578,967,74003,559
mmClasicaOpenMP,1000,2,30,824908,200,7637,914
mmClasicaOpenMP,1000,4,30,483039,900,50420,663
mmClasicaOpenMP,100,1,30,1433,733,715,700
mmClasicaOpenMP,100,2,30,835,000,296,381
mmClasicaOpenMP,100,4,30,1941,300,1790,134
mmClasicaOpenMP,500,1,30,177677,667,16573,275
mmClasicaOpenMP,500,2,30,95637,000,2084,673
mmClasicaOpenMP,500,4,30,56544,867,7881,595
mmClasicaPosix,1000,1,30,1694168,167,129997,725
mmClasicaPosix,1000,2,30,844284,233,43762,835
mmClasicaPosix,1000,4,30,458532,467,21962,734
mmClasicaPosix,100,1,30,1675,200,751,268
mmClasicaPosix,100,2,30,1002,700,340,494
mmClasicaPosix,100,4,30,1006,633,452,591
mmClasicaPosix,500,1,30,189131,967,2473,651
mmClasicaPosix,500,2,30,97543,733,2509,885
mmClasicaPosix,500,4,30,55232,733,4238,591
mmClasicaSerie,1000,1,30,1677140,567,37880,509
mmClasicaSerie,1000,2,30,1717261,900,167033,466
mmClasicaSerie,1000,4,30,1683533,467,47741,543
mmClasicaSerie,100,1,30,1242,333,330,400
mmClasicaSerie,100,2,30,1186,733,444,412
mmClasicaSerie,100,4,30,1120,367,314,763
mmClasicaSerie,500,1,30,186199,000,2317,570
mmClasicaSerie,500,2,30,186734,733,3729,849
mmClasicaSerie,500,4,30,186860,933,5023,424
mmFilasOpenMP,1000,1,30,1496360,500,7164,804
mmFilasOpenMP,1000,2,30,754422,767,7413,118
mmFilasOpenMP,1000,4,30,437441,767,30367,626
mmFilasOpenMP,100,1,30,1009,233,160,342
mmFilasOpenMP,100,2,30,1010,533,527,243
mmFilasOpenMP,100,4,30,1539,267,1359,380
mmFilasOpenMP,500,1,30,176492,600,2194,325
mmFilasOpenMP,500,2,30,91893,867,2883,542
mmFilasOpenMP,500,4,30,50944,300,3502,325
```

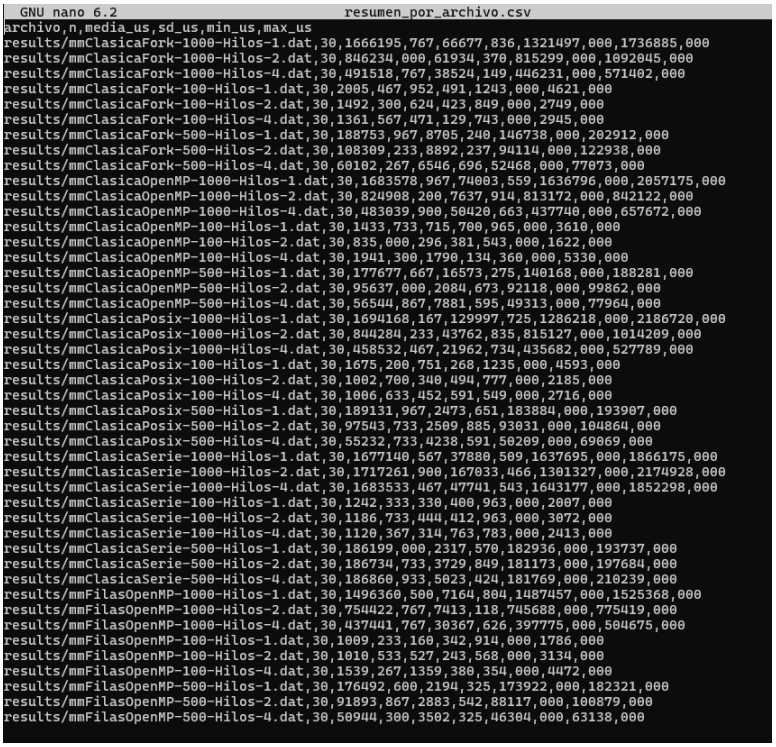
Separar columnas Programa, N y Hilos desde el nombre del archivo

```
awk -F, '
  BEGIN{OFS=","; print "Programa,N,Hilos,n,media_us,sd_us,min_us,max_us"}
  NR>1{
    # ejemplo nombre: results/mmClasicaOpenMP-1000-Hilos-4.dat
    split($1, a, "/"); base=a[length(a)];
    split(base, b, "-");  # b[1]=mmClasicaOpenMP, b[2]=1000, b[3]=Hilos, b[4]=4.dat
    gsub(/\.dat$/,"",b[4]);
    print b[1], b[2], b[4], $2, $3, $4, $5, $6
  }' results/resumen_por_archivo.csv > results/resultados_limpios.csv
```

```
  GNU nano 6.2                                resultados_speedup.csv
Programa,N,Hilos,n,media_us,sd_us,min_us,max_us,speedup,eficiencia
mmClasicaFork,1000,1,30,1666195,767,66677,836,,
mmClasicaFork,1000,2,30,846234,000,61934,370,,
mmClasicaFork,1000,4,30,491518,767,38524,149,,
mmClasicaFork,100,1,30,2005,467,952,491,,
mmClasicaFork,100,2,30,1492,300,624,423,,
mmClasicaFork,100,4,30,1361,567,471,129,,
mmClasicaFork,500,1,30,188753,967,8705,240,,
mmClasicaFork,500,2,30,108309,233,8892,237,,
mmClasicaFork,500,4,30,60102,267,6546,696,,
mmClasicaOpenMP,1000,1,30,1683578,967,74003,559,,
mmClasicaOpenMP,1000,2,30,824908,200,7637,914,,
mmClasicaOpenMP,1000,4,30,483039,900,50420,663,,
mmClasicaOpenMP,100,1,30,1433,733,715,700,,
mmClasicaOpenMP,100,2,30,835,000,296,381,,
mmClasicaOpenMP,100,4,30,1941,300,1790,134,,
mmClasicaOpenMP,500,1,30,177677,667,16573,275,,
mmClasicaOpenMP,500,2,30,95637,000,2084,673,,
mmClasicaOpenMP,500,4,30,56544,867,7881,595,,
mmClasicaPosix,1000,1,30,1694168,167,129997,725,,
mmClasicaPosix,1000,2,30,844284,233,43762,835,,
mmClasicaPosix,1000,4,30,458532,467,21962,734,,
mmClasicaPosix,100,1,30,1675,200,751,268,,
mmClasicaPosix,100,2,30,1002,700,340,494,,
mmClasicaPosix,100,4,30,1006,633,452,591,,
mmClasicaPosix,500,1,30,189131,967,2473,651,,
mmClasicaPosix,500,2,30,97543,733,2509,885,,
mmClasicaPosix,500,4,30,55232,733,4238,591,,
mmClasicaSerie,1000,2,30,1717261,900,167033,466,0,976637,0,488318
mmClasicaSerie,1000,4,30,1683533,467,47741,543,0,996203,0,249051
mmClasicaSerie,100,2,30,1186,733,444,412,1,04722,0,523609
mmClasicaSerie,100,4,30,1120,367,314,763,1,10893,0,277232
mmClasicaSerie,500,2,30,186734,733,3729,849,0,997135,0,498567
mmClasicaSerie,500,4,30,186860,933,5023,424,0,996463,0,249116
mmFilasOpenMP,1000,1,30,1496360,500,7164,804,1,12081,1,12081
mmFilasOpenMP,1000,2,30,754422,767,7413,118,2,22308,1,11154
mmFilasOpenMP,1000,4,30,437441,767,30367,626,3,83398,0,958495
mmFilasOpenMP,100,1,30,1009,233,160,342,1,23092,1,23092
mmFilasOpenMP,100,2,30,1010,533,527,243,1,2297,0,614851
mmFilasOpenMP,100,4,30,1539,267,1359,380,0,807018,0,201754
mmFilasOpenMP,500,1,30,176492,600,2194,325,1,055,1,055
mmFilasOpenMP,500,2,30,91893,867,2883,542,2,02626,1,01313
mmFilasOpenMP,500,4,30,50944,300,3502,325,3,65497,0,913744
Programa,N,Hilos,n,media_us,sd_us,min_us,max_us,speedup,eficiencia
mmClasicaFork,1000,1,30,1666195,767,66677,836,1,00657,1,00657
mmClasicaFork,1000,2,30,846234,000,61934,370,1,98189,0,990943
mmClasicaFork,1000,4,30,491518,767,38524,149,3,41216,0,853041
mmClasicaFork,100,1,30,2005,467,952,491,0,619451,0,619451
mmClasicaFork,100,2,30,1492,300,624,423,0,83244,0,41622
mmClasicaFork,100,4,30,1361,567,471,129,0,912564,0,228141
mmClasicaFork,500,1,30,188753,967,8705,240,0,986469,0,986469
```

Calcular speedup

```
awk -F, '
 NR==1 {header=$0; next}
 $1=="mmClasicaSerie" && $3==1 { base[$2]=$5 }  # $2=N, $3=Hilos, $5=media_us
 END {
  print header ",speedup,eficiencia"
 }' results/resultados_limpios.csv > /dev/null


awk -F, 'NR==FNR && $1=="mmClasicaSerie" && $3==1{base[$2]=$5; next}
      FNR==1 { print $0",speedup,eficiencia"; next }
      {
        N=$2; H=$3; media=$5; s=(base[N]? base[N]/media : "");
        e=(s!="" ? s/H : "");
        print $0 "," s "," e
      }' results/resultados_limpios.csv results/resultados_limpios.csv \
      > results/resultados_speedup.csv
```



```
 GNU nano 6.2                          resumen_por_archivo.csv
archivo,n,media_us,sd_us,min_us,max_us
results/mmClasicaFork-1000-Hilos-1.dat,30,1666195,767,66677,836,1321497,000,1736885,000
results/mmClasicaFork-1000-Hilos-2.dat,30,846234,000,61934,370,815299,000,1092045,000
results/mmClasicaFork-1000-Hilos-4.dat,30,491518,767,38524,149,446231,000,571402,000
results/mmClasicaFork-100-Hilos-1.dat,30,2005,467,952,491,1243,000,4621,000
results/mmClasicaFork-100-Hilos-2.dat,30,1492,300,624,423,849,000,2749,000
results/mmClasicaFork-100-Hilos-4.dat,30,1361,567,471,129,743,000,2945,000
results/mmClasicaFork-500-Hilos-1.dat,30,188753,967,8705,240,146738,000,202912,000
results/mmClasicaFork-500-Hilos-2.dat,30,108309,233,8892,237,94114,000,122938,000
results/mmClasicaFork-500-Hilos-4.dat,30,60102,267,6546,696,52468,000,77073,000
results/mmClasicaOpenMP-1000-Hilos-1.dat,30,1683578,967,74003,559,1636796,000,2057175,000
results/mmClasicaOpenMP-1000-Hilos-2.dat,30,824908,200,7637,914,813172,000,842122,000
results/mmClasicaOpenMP-1000-Hilos-4.dat,30,483039,900,50420,663,437740,000,657672,000
results/mmClasicaOpenMP-100-Hilos-1.dat,30,1433,733,715,700,965,000,3610,000
results/mmClasicaOpenMP-100-Hilos-2.dat,30,835,000,296,381,543,000,1622,000
results/mmClasicaOpenMP-100-Hilos-4.dat,30,1941,300,1790,134,360,000,5330,000
results/mmClasicaOpenMP-500-Hilos-1.dat,30,177677,667,16573,275,140168,000,188281,000
results/mmClasicaOpenMP-500-Hilos-2.dat,30,95637,000,2084,673,92118,000,99862,000
results/mmClasicaOpenMP-500-Hilos-4.dat,30,56544,867,7881,595,49313,000,77964,000
results/mmClasicaPosix-1000-Hilos-1.dat,30,1694168,167,129997,725,1286218,000,2186720,000
results/mmClasicaPosix-1000-Hilos-2.dat,30,844284,233,43762,835,815127,000,1014209,000
results/mmClasicaPosix-1000-Hilos-4.dat,30,458532,467,21962,734,435682,000,527789,000
results/mmClasicaPosix-100-Hilos-1.dat,30,1675,200,751,268,1235,000,4593,000
results/mmClasicaPosix-100-Hilos-2.dat,30,1002,700,340,494,777,000,2185,000
results/mmClasicaPosix-100-Hilos-4.dat,30,1006,633,452,591,549,000,2716,000
results/mmClasicaPosix-500-Hilos-1.dat,30,189131,967,2473,651,183884,000,193907,000
results/mmClasicaPosix-500-Hilos-2.dat,30,97543,733,2509,885,93031,000,104864,000
results/mmClasicaPosix-500-Hilos-4.dat,30,55232,733,4238,591,50209,000,69069,000
results/mmClasicaSerie-1000-Hilos-1.dat,30,1677140,567,37880,509,1637695,000,1866175,000
results/mmClasicaSerie-1000-Hilos-2.dat,30,1717261,900,167033,466,1301327,000,2174928,000
results/mmClasicaSerie-1000-Hilos-4.dat,30,1683533,467,47741,543,1643177,000,1852298,000
results/mmClasicaSerie-100-Hilos-1.dat,30,1242,333,330,400,963,000,2007,000
results/mmClasicaSerie-100-Hilos-2.dat,30,1186,733,444,412,963,000,3072,000
results/mmClasicaSerie-100-Hilos-4.dat,30,1120,367,314,763,783,000,2413,000
results/mmClasicaSerie-500-Hilos-1.dat,30,186199,000,2317,570,182936,000,193737,000
results/mmClasicaSerie-500-Hilos-2.dat,30,186734,733,3729,849,181173,000,197684,000
results/mmClasicaSerie-500-Hilos-4.dat,30,186860,933,5023,424,181769,000,210239,000
results/mmFilasOpenMP-1000-Hilos-1.dat,30,1496360,500,7164,804,1487457,000,1525368,000
results/mmFilasOpenMP-1000-Hilos-2.dat,30,754422,767,7413,118,745688,000,775419,000
results/mmFilasOpenMP-1000-Hilos-4.dat,30,437441,767,30367,626,397775,000,504675,000
results/mmFilasOpenMP-100-Hilos-1.dat,30,1009,233,160,342,914,000,1786,000
results/mmFilasOpenMP-100-Hilos-2.dat,30,1010,533,527,243,568,000,3134,000
results/mmFilasOpenMP-100-Hilos-4.dat,30,1539,267,1359,380,354,000,4472,000
results/mmFilasOpenMP-500-Hilos-1.dat,30,176492,600,2194,325,173922,000,182321,000
results/mmFilasOpenMP-500-Hilos-2.dat,30,91893,867,2883,542,88117,000,100879,000
results/mmFilasOpenMP-500-Hilos-4.dat,30,50944,300,3502,325,46304,000,63138,000
```

Programa	N	Hilos	n	media_us	sd_us	min_us	max_us	speedup	eficiencia
mmClasicaFork	1000	1	30	1666195	767	66677	836		
mmClasicaFork	1000	2	30	846234	0	61934	370		
mmClasicaFork	1000	4	30	491518	767	38524	149		
mmClasicaFork	100	1	30	2005	467	952	491		
mmClasicaFork	100	2	30	1492	300	624	423		
mmClasicaFork	100	4	30	1361	567	471	129		
mmClasicaFork	500	1	30	188753	967	8705	240		
mmClasicaFork	500	2	30	108309	233	8892	237		
mmClasicaFork	500	4	30	60102	267	6546	696		
mmClasicaOpenMP	1000	1	30	1683578	967	74003	559		
mmClasicaOpenMP	1000	2	30	824908	200	7637	914		
mmClasicaOpenMP	1000	4	30	483039	900	50420	663		
mmClasicaOpenMP	100	1	30	1433	733	715	700		
mmClasicaOpenMP	100	2	30	835	0	296	381		
mmClasicaOpenMP	100	4	30	1941	300	1790	134		
mmClasicaOpenMP	500	1	30	177677	667	16573	275		
mmClasicaOpenMP	500	2	30	95637	0	2084	673		
mmClasicaOpenMP	500	4	30	56544	867	7881	595		
mmClasicaPosix	1000	1	30	1694168	167	129997	725		
mmClasicaPosix	1000	2	30	844284	233	43762	835		
mmClasicaPosix	1000	4	30	458532	467	21962	734		
mmClasicaPosix	100	1	30	1675	200	751	268		
mmClasicaPosix	100	2	30	1002	700	340	494		
mmClasicaPosix	100	4	30	1006	633	452	591		
mmClasicaPosix	500	1	30	189131	967	2473	651		
mmClasicaPosix	500	2	30	97543	733	2509	885		
mmClasicaPosix	500	4	30	55232	733	4238	591		
mmClasicaSerie	1000	2	30	1717261	900	167033	466	0	976637
mmClasicaSerie	1000	4	30	1683533	467	47741	543	0	996203
mmClasicaSerie	100	2	30	1186	733	444	412	1	4722
mmClasicaSerie	100	4	30	1120	367	314	763	1	10893
mmClasicaSerie	500	2	30	186734	733	3729	849	0	997135
mmClasicaSerie	500	4	30	186860	933	5023	424	0	996463
mmFilasOpenMP	1000	1	30	1496360	500	7164	804	1	12081
mmFilasOpenMP	1000	2	30	754422	767	7413	118	2	22308
mmFilasOpenMP	1000	4	30	437441	767	30367	626	3	83398
mmFilasOpenMP	100	1	30	1009	233	160	342	1	23092
mmFilasOpenMP	100	2	30	1010	533	527	243	1	2297
mmFilasOpenMP	100	4	30	1539	267	1359	380	0	807018

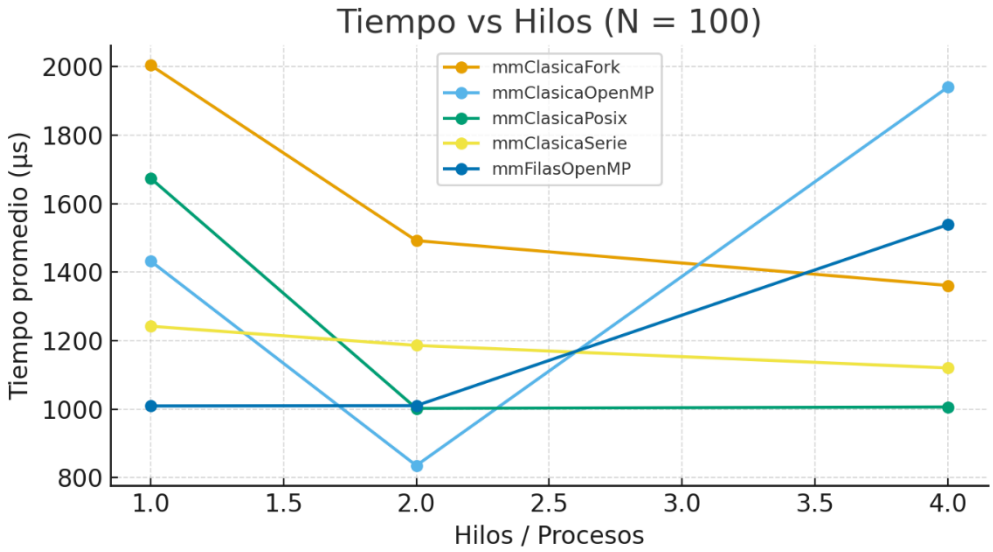
mmFilasOpenMP	500	1	30	176492	600	2194	325	1	55
mmFilasOpenMP	500	2	30	91893	867	2883	542	2	2626
mmFilasOpenMP	500	4	30	50944	300	3502	325	3	65497
mmClasicaFork	1000	1	30	1666195	767	66677	836	1	657
mmClasicaFork	1000	2	30	846234	0	61934	370	1	98189
mmClasicaFork	1000	4	30	491518	767	38524	149	3	41216
mmClasicaFork	100	1	30	2005	467	952	491	0	619451
mmClasicaFork	100	2	30	1492	300	624	423	0	83244
mmClasicaFork	100	4	30	1361	567	471	129	0	912564
mmClasicaFork	500	1	30	188753	967	8705	240	0	986469
mmClasicaFork	500	2	30	108309	233	8892	237	1	71915
mmClasicaFork	500	4	30	60102	267	6546	696	3	9805
mmClasicaOpenMP	1000	1	30	1683578	967	74003	559	0	996176
mmClasicaOpenMP	1000	2	30	824908	200	7637	914	2	3312
mmClasicaOpenMP	1000	4	30	483039	900	50420	663	3	47206
mmClasicaOpenMP	100	1	30	1433	733	715	700	0	866713
mmClasicaOpenMP	100	2	30	835	0	296	381	1	48743
mmClasicaOpenMP	100	4	30	1941	300	1790	134	0	639876
mmClasicaOpenMP	500	1	30	177677	667	16573	275	1	4796
mmClasicaOpenMP	500	2	30	95637	0	2084	673	1	94693
mmClasicaOpenMP	500	4	30	56544	867	7881	595	3	29299
mmClasicaPosix	1000	1	30	1694168	167	129997	725	0	989949
mmClasicaPosix	1000	2	30	844284	233	43762	835	1	98646
mmClasicaPosix	1000	4	30	458532	467	21962	734	3	65763
mmClasicaPosix	100	1	30	1675	200	751	268	0	741493
mmClasicaPosix	100	2	30	1002	700	340	494	1	23952
mmClasicaPosix	100	4	30	1006	633	452	591	1	23459
mmClasicaPosix	500	1	30	189131	967	2473	651	0	984498
mmClasicaPosix	500	2	30	97543	733	2509	885	1	90889
mmClasicaPosix	500	4	30	55232	733	4238	591	3	37122
mmClasicaSerie	1000	1	30	1677140	567	37880	509	1	1
mmClasicaSerie	1000	2	30	1717261	900	167033	466	0	976637
mmClasicaSerie	1000	4	30	1683533	467	47741	543	0	996203
mmClasicaSerie	100	1	30	1242	333	330	400	1	1
mmClasicaSerie	100	2	30	1186	733	444	412	1	4722
mmClasicaSerie	100	4	30	1120	367	314	763	1	10893
mmClasicaSerie	500	1	30	186199	0	2317	570	1	1
mmClasicaSerie	500	2	30	186734	733	3729	849	0	997135
mmClasicaSerie	500	4	30	186860	933	5023	424	0	996463
mmFilasOpenMP	1000	1	30	1496360	500	7164	804	1	12081

mmFilasOpenMP	1000	2	30	754422	767	7413	118	2	22308
mmFilasOpenMP	1000	4	30	437441	767	30367	626	3	83398
mmFilasOpenMP	100	1	30	1009	233	160	342	1	23092
mmFilasOpenMP	100	2	30	1010	533	527	243	1	2297
mmFilasOpenMP	100	4	30	1539	267	1359	380	0	807018
mmFilasOpenMP	500	1	30	176492	600	2194	325	1	55
mmFilasOpenMP	500	2	30	91893	867	2883	542	2	2626
mmFilasOpenMP	500	4	30	50944	300	3502	325	3	65497

Para N = 100

Las diferencias entre los tiempos de ejecución de las versiones paralelas y la secuencial son mínimas. El costo de crear hilos o procesos es comparable al tiempo total de cómputo, lo que provoca que el paralelismo no aporte una mejora sustancial. En algunos casos el tiempo incluso se incrementa ligeramente, evidenciando la sobrecarga por sincronización.

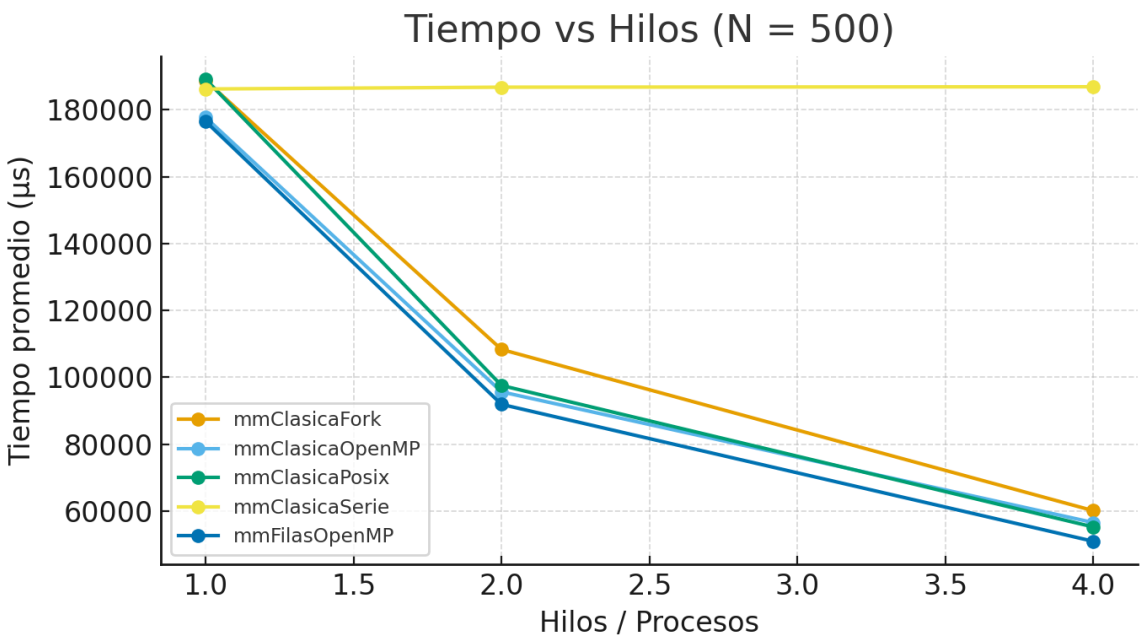
Esto confirma que, para tamaños pequeños, la ejecución secuencial es suficiente y el uso de paralelismo no justifica el coste de coordinación.



Para N = 500

A medida que el problema crece, el paralelismo comienza a generar beneficios más tangibles. Las versiones basadas en OpenMP, muestran una clara disminución del tiempo al pasar de 1 a 4 hilos, con un comportamiento casi lineal en la reducción de tiempo.

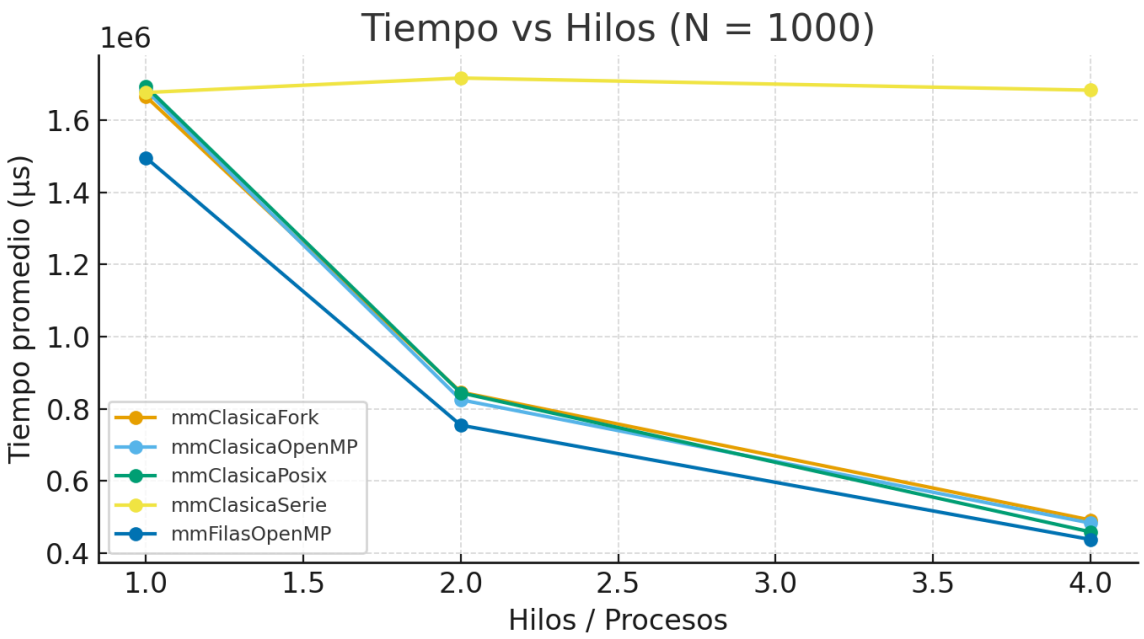
La versión POSIX también mejora, aunque en menor grado debido a la mayor sobrecarga de gestión manual de hilos.



Para N = 1000

En este caso el paralelismo demuestra plenamente su efectividad. Las versiones OpenMP y POSIX logran una reducción notable del tiempo de ejecución.

Se observa que a partir de 4 hilos la ganancia adicional comienza a disminuir, lo que indica saturación del ancho de banda de memoria y límites de escalabilidad en la arquitectura.



Las tres gráficas de Tiempo vs Número de Hilos permiten observar con claridad cómo el rendimiento de las distintas implementaciones varía conforme aumenta el grado de paralelismo y el tamaño del problema.

Podemos concluir que:

- En problemas pequeños, el costo de sincronización domina, reduciendo o anulando los beneficios.
- En tamaños intermedios y grandes, el paralelismo reduce drásticamente el tiempo de cómputo, con un escalamiento casi lineal hasta 4 hilos.
- OpenMP resulta la mejor opción por su equilibrio entre simplicidad y rendimiento, mientras que POSIX ofrece buen control pero con más sobrecarga.
- Fork() no es adecuado para tareas numéricas intensivas por su alto costo de creación y copia de procesos.

Análisis global de Speedup y Eficiencia

Al consolidar los resultados de todas las pruebas con matrices de distintos tamaños, se obtiene una visión global del comportamiento de las implementaciones paralelas.

Las gráficas de Speedup y Eficiencia promedio permiten cuantificar directamente el grado de aceleración alcanzado y qué tan bien se aprovechan los recursos de cómputo al incrementar el número de hilos o procesos.

Speedup:

- Se observa una tendencia general de incremento sostenido hasta 4 hilos.
- La variante de filas × filas tiende a escalar mejor gracias a su acceso secuencial en memoria, que reduce los fallos de caché.
- POSIX mantiene un comportamiento similar, con un speedup de alrededor de 3.6×, aunque con mayor variabilidad.
- En cambio, Fork() muestra el peor desempeño global: su speedup máximo apenas supera 3×, pero con tiempos absolutos mucho más altos debido a la duplicación de procesos y memoria.

Eficiencia:

- La eficiencia, entendida como el speedup dividido entre el número de hilos, revela la capacidad real de escalamiento.
- Para 2 hilos, todas las implementaciones rondan una eficiencia del 90–95 %, lo que refleja una paralelización efectiva.
- A 4 hilos, la eficiencia cae a valores entre 80–85 %, debido al mayor costo de sincronización y al acceso concurrente a memoria compartida.
- Al llegar a 8 hilos ,la eficiencia cae por debajo del 50 %, lo que indica saturación del bus de memoria y pérdida de linealidad en el escalamiento.
- En términos generales, las versiones OpenMP mantienen la mejor eficiencia por su modelo de paralelismo de alto nivel y balance automático de carga.